Reward-based learning under hardware constraints - Using a RISC processor embedded in a neuromorphic substrate

Simon Friedmann[*1], **Nicolas Frémaux**[2], **Johannes Schemmel**[1], **Wulfram Gerstner**[2], **Karlheinz Meier**[1]

[1] Kirchhoff Institute for Physics, Ruprecht-Karls-University Heidelberg, Germany
[2] School of Computer and Communication Sciences and Brain-Mind Institute, Ecole Polytechnique Fédérale de Lausanne, Switzerland

[*]**Correspondence:**
Simon Friedmann
Ruprecht-Karls-University Heidelberg
Kirchhoff Institute for Physics
Im Neuenheimer Feld 227
69120 Heidelberg, Germany
simon.friedmann@kip.uni-heidelberg.de

Abstract

In this study, we propose and analyze in simulations a new, highly flexible method of implementing synaptic plasticity in a wafer-scale, accelerated neuromorphic hardware system. The study focuses on globally modulated STDP, as a special use-case of this method. Flexibility is achieved by embedding a general-purpose processor dedicated to plasticity into the wafer. To evaluate the suitability of the proposed system, we use a reward modulated STDP rule in a spike train learning task. A single layer of neurons is trained to fire at specific points in time with only the reward as feedback. This model is simulated to measure its performance, i.e. the increase in received reward after learning. Using this performance as baseline, we then simulate the model with various constraints imposed by the proposed implementation and compare the performance. The simulated constraints include discretized synaptic weights, a restricted interface between analog synapses and embedded processor, and mismatch of analog circuits. We find that probabilistic updates can increase the performance of low-resolution weights, a simple interface between analog synapses and processor is sufficient for learning, and performance is insensitive to mismatch. Further, we consider communication latency between wafer and the conventional control computer system that is simulating the environment. This latency increases the delay, with which the reward is sent to the embedded processor. Because of the time continuous operation of the analog synapses, delay can cause a deviation of the updates as compared to the not delayed situation. We find that for highly accelerated systems latency has to be kept to a minimum. This study shows the proposed implementation to be suitable to model reward modulated STDP learning rules. It is therefore an ideal candidate for implementation in an upgraded version of the wafer-scale system developed within the BrainScaleS project.

Keywords: neuromorphic hardware, wafer-scale integration, large-scale spiking neural networks, spike-timing dependent plasticity, reinforcement learning, hardware constraints analysis

1. Introduction

In reinforcement learning, an agent learns to achieve a goal through interaction with an environment (Sutton and Barto, 1998). The environment provides a single scalar number, the reward, as feedback to the actions performed by the learning agent. The agent tries to maximize the reward it receives over time by changing its behavior. In contrast to supervised learning, where an instructor supplies the correct actions to take, here the agent has to learn the correct strategy itself through trial-and-error. Typically this is done by introducing randomness in the selection of actions and taking into account the resulting reward. Recently, several studies have suggested extending classical spike-timing dependent plasticity (STDP, Morrison et al., 2008; Caporale and Dan, 2008) into reward-modulated STDP to implement reinforcement learning in the context of spiking neural networks (Izhikevich, 2007; Farries and Fairhall, 2007; Florian, 2007; Legenstein et al., 2008; Fremaux et al., 2010; Potjans et al., 2011). One of the key issues in reinforcement learning is solving the so-called temporal credit assignment problem: reward arrives some time after the action that caused it. So how does the agent know how to change its behavior? It needs to retain some information about recent actions in order to assign proper credit for the rewards it receives. To do this, reward modulated STDP generates an eligibility trace for every synapse that depends on pre- and postsynaptic firing. This trace, modulated by the reward, determines the change of synaptic weight, thereby solving the credit assignment problem.

Spike-based implementations do not only offer an approach to biological models of learning, they are also suitable for implementation in neuromorphic hardware devices. Existing systems offer a number of interesting characteristics, such as low-power consumption (e.g. Wijekoon and Dudek, 2008; Livi and Indiveri, 2009; Seo et al., 2011), faster than real-time dynamics (Wijekoon and Dudek, 2008; Schemmel et al., 2010), and scalability to large networks (Schemmel et al., 2010; Furber et al., 2012). They are typically built with two goals in mind: as new kind of brain inspired information processing device and to provide a scalable platform for the experimental exploration of networks. Currently, learning capabilities are limited to variants of unsupervised STDP (Indiveri et al., 2006; Schemmel et al., 2006; Seo et al., 2011; Ramakrishnan et al., 2011; Davies et al., 2012).

In this study we analyze the implementability of a reward-modulated STDP model derived from Fremaux et al. (2010) in neuromorphic hardware. To that end, we propose an extended version of the BrainScaleS wafer-scale system (Schemmel et al., 2008; Fieres et al., 2008; Schemmel et al., 2010) to serve as a conceptual basis for this analysis. This system is designed as a power-efficient, faster than real-time and flexible emulation platform for large neural networks. In particular the acceleration in time compared to biology make it interesting for reinforcement learning, which typically suffers from slow convergence (Sutton and Barto, 1998). Starting from an existing system with limited modifications leads to a more realistic design prototype compared to starting from scratch.

A key objective for the proposed neuromorphic system is to be a valuable tool for neuroscience. Therefore, the design must not be targeted at a single network architecture, task or learning rule, but instead stay as flexible as is reasonably possible. On the other hand, implementing large-scale neural networks with accelerated time-scale raises technical challenges and trade-offs have to be made between flexibility and performance. The proposed extension rep-

resents a plasticity mechanism reflecting this design philosophy: specialized analog circuits in every synapse are combined with a general purpose embedded plasticity-processor (EPP). This way, the benefits from the worlds of analog and processor-based computing can be combined: analog circuits are used for compact, power-efficient and fast local processing, and digital processors allow for programmable plasticity rules. Integrating the processors into the same application specific integrated circuits (ASIC) on the wafer as the neuromorphic substrate allows for scalability to wafer size networks and beyond.

In the following, we will consider only the aforementioned rule studied in Fremaux et al. (2010) and analyze effects caused by the adaptation to the hardware system in simulations. We want to answer the question whether the hybrid approach combining processor and analog circuits is a suitable platform for this particular learning rule. Among the hardware-induced constraints are non-continuous weights, drift of analog circuits and communication latency between hardware substrate and the controlling computer system. We want to test and compare the performance of the unconstrained and the constrained plasticity rules in order to find guidelines for the hardware implementation, for example the required weight resolution or maximum noise levels. Section 2 describes the extended hardware system and the plasticity model. Section 3 presents results from simulations showing performance under hardware constraints. Section 4 provides a discussion of our results.

2. Materials and methods

2.1. Using an embedded processor for plasticity

The key concept of our hardware implementation of synaptic plasticity is to use a programmable general-purpose processor in combination with fixed-function analog hardware. Software running on the processor can use observables and controls to interface with the neuromorphic substrate. Thereby, it is possible to flexibly switch between synaptic learning rules or use different ones in parallel for different synapses. The alternative to this concept would be to use fixed-function hardware instead of a general-purpose processor. This would allow a more efficient implementation of one specific rule, at the cost of system versatility. In the following, we give background information on a complete neuromorphic system following the concept of processor-enabled plasticity. From the system described, we derive hardware constraints that are used in the simulations reported in Section 3.

2.1.1. System overview

Figure 1 gives a schematic overview of the complete hardware system. The experimenter controls the system through a control cluster of off-the-shelf computers. The network is provided in a description abstracted from the details of the system using the PyNN modelling language (Davison et al., 2008). An automated mapping process translates the description into the detailed configuration that is written to the wafer-modules (Ehrlich et al., 2010; Wendt et al., 2008). These modules are interconnected by a high-speed network to communicate spike-events (Scholze et al.,

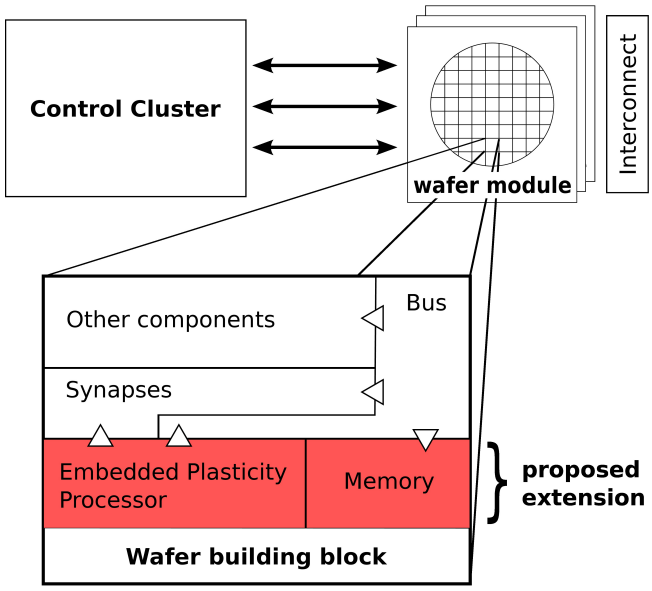

Figure 1: Overview of the system. The user controls the system through a cluster of conventional computers by sending configuration and spike data to a number of modules that each carry a wafer. These wafer modules are interconnected with a high-speed network to exchange spike events. The wafer contains identical building blocks, of which one is shown in an expanded view. The proposed extension to the BrainScaleS wafer-scale system in form of the embedded plasticity processor is marked in red. Input/output access from the processor to other components of the building block is indicated with triangles.

2011). External stimulation can be applied to the network from the control cluster, using the high-speed links that are also used for configuration. The wafer itself is subdivided into building blocks that contain the neuromorphic substrate, i.e. synapses, neurons, parameter storage and networking resources for spike transmission.

Our proposed extension adds an embedded plasticity processor (EPP) to every building block on the wafer, together with its own memory for instructions and data. It will be equipped with three interfaces to the fixed-function hardware: read and write access on the synapses, rate counters and event generation for the network and access to the control bus of the building block. The latter is also used by external control accesses and thus, a plasticity program running on the embedded processor will be able to do everything that could be done from an off-wafer control computer as long as it only requires information local to the block. There is no direct communication channel between processors envisioned, but software on the control computer could be used for data exchange.

2.1.2. Implementing plasticity

Our proposed design represents a hybrid system, in which the digital EPP interacts closely with analog components. Every synapse contains an analog accumulation circuit, similar to the version used in an earlier design (Schemmel et al., 2007). For each pre-post and post-pre spike-pair, the time difference Δt is measured and weighted exponentially using the amplitude $A_\pm$ and time constant $\tau_\pm$:

$$\delta_\pm = A_\pm \exp\left(-\frac{|\Delta t|}{\tau_\pm}\right). \tag{1}$$

These values are added to two local capacitors a_+ and a_-, respectively. In the extended version the EPP will select synapses for readout and use an analog evaluation unit to produce a series of bits b_i out of a_+ and a_-. The evaluation function can perform different readout operations controlled by configuration bits e^i_{cc}, e^i_{ca}, e^i_{ac} and e^i_{aa} and analog parameters a_{tl} and a_{th}:

$$b_i = \begin{cases} 1 & \text{if } \frac{a_{tl}+e^i_{ac}a_+ + e^i_{ca}a_-}{1+e^i_{ac}+e^i_{ca}} > \frac{a_{th}+e^i_{cc}a_+ + e^i_{aa}a_-}{1+e^i_{cc}+e^i_{aa}} \\ 0 & \text{otherwise} \end{cases}. \tag{2}$$

Using $b_0 \dots b_{N-1}$, the current weight of the synapse w and possibly further global parameters $P_0 \dots P_{M-1}$ as input, the weight update Δ is then calculated in software by the EPP:

$$\Delta = F\left(b_0, \dots, b_{N-1}, w, P_0, \dots, P_{M-1}\right) \tag{3}$$

Then, the new weight $w' = w + \Delta$ is written to weight storage by the plasticity program. Using two evaluations b_0, b_1 with different sets of configuration bits, a simple example for F would be:

$$F\left(b_0, b_1\right) = \tilde{A}_0 b_0 + \tilde{A}_1 b_1 \tag{4}$$

With arbitrary constants $\tilde{A}_0$ and $\tilde{A}_1$.

Synapses in the system are organized in an array of synapse-units, where each synapse has a 4 bit weight memory implemented with static random-access memory (SRAM) cells. These

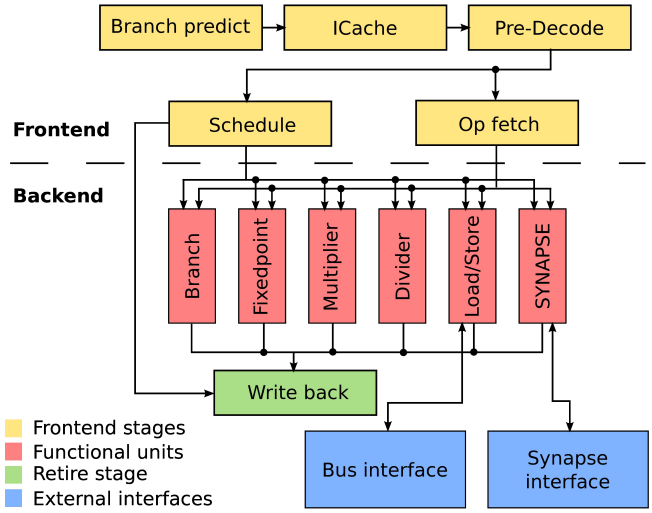

Figure 2: Micro-architecture of the embedded plasticity processor. The design is separated into frontend and backend. The frontend takes four clock cycles to decode instructions and issue them in-order to the applicable functional unit. The functional units take a minimum of two cycles. Writing the result back to the register file takes another cycle. Input/output operations are performed through a bus interface served by the load/store unit and a specialized interface to the synapse array.

offer the ability to combine adjacent units to increase resolution to 8 bit. Of course this has the negative effect of reducing the total amount of implementable synapses.

2.1.3. Embedded micro-processor

Plasticity algorithms will be implemented by software programs executed on the EPP. A large class of micro-processors is in use today for various different applications from supercomputers, to smartphones and embedded controllers for traffic lights. They all use different computer architectures reflecting the specific requirements and constraints of their application.

There are three important characteristics for a processor: one, the used instruction set architecture (ISA) that defines coding and semantics of instructions and registers. Two, whether instructions are executed out-of-order and three, whether the design is super-scalar, i.e. instructions can execute in parallel. The instruction set architecture used here is a subset of the PowerISA 2.06 specification for 32 bit (PowerISA, 2010). The main reason to use an existing ISA is the availability of compilers and tools. Code for the EPP can be generated using the GNU Compiler Collection (Stallman, 2012), using the C programming language.

The micro-architecture of the EPP is shown in Figure 2. The frontend fetches and issues instruction in program order to the functional units. Due to different latencies, instructions can retire out of program order to the write back stage. For example a slow memory access may be overtaken by a quick add instruction issued after it. Program and data are stored in a 12 kiB memory. A direct-mapped cache (*ICache*) is used for instruction access and to avoid the von-

Neumann bottleneck (Backus, 1978). Branches can be predicted with a fully associative branch predictor using 2 bit saturating counters to track branch outcome (Strategy 7 in Smith, 1998). The functional units include load/store for memory access, a branch facility for control transfers, fixed-point arithmetic and logical instructions including a barrel shifter, multiply and divide. The SYNAPSE special-function unit implements application specific instructions and registers. It allows for accelerated weight computation and synapse access.

An important goal for our proposed design is to maintain small area requirements to allow integration into the existing BrainScaleS wafer-scale system. To this end, we chose in-order issue of instructions to avoid additional control logic associated with tracking of instructions and reordering. However, out-of-order completion can be achieved with relatively small area overhead using a result shift-register (Smith and Pleszkun, 1985) and was therefore included to improve performance.

2.2. Model for reinforcement learning

To demonstrate reinforcement learning using the proposed system architecture, we chose a plasticity rule and a learning task described in Fremaux et al. (2010). The R-STDP rule (Izhikevich, 2007; Florian, 2007) is a three-factor synaptic plasticity learning rule that modulates classical two-factor STDP with a reward-based success signal S. At the end of each trial of the learning task, a reward R is calculated according to the performance of the network and is used to modify the weights according to the learning rule.

2.2.1. Network model

The network we simulate consists of two layers, connected with plastic synapses using the reward-modulated learning rule. The input layer consists of units repeating a given set of spike trains. The output layer consists of spiking neurons, being excited by the fixed activity from the input layer.

The original network in Fremaux et al. (2010) uses the simplified Spike Response Model (SRM_0, Gerstner and Kistler, 2002) for the output neurons. It is an intrinsically stochastic neuron that emits spikes based on the exponentially weighted distance to the threshold. In hardware the most commonly used neuron type is the deterministic leaky integrate-and-fire (LIF). The proposed system would use the hardware neuron reported in Millner et al. (2010) that can be operated as Adaptive Exponential Integrate-and-Fire (AdEx, Brette and Gerstner, 2005) or conventional LIF model. Since a certain amount of randomness in the firing behavior is required for reinforcement learning, we add background noise stimulation in the form of Poisson processes.

A tabular description of the network model can be found in Table 1. N_U input units project onto N_T neurons that are additionally stimulated by N_B random background sources. All neurons are connected to all inputs, but each has individual random stimulation from equally sized and disjoint subsets of the random background. In every trial the same input spike pattern is presented, but the background noise realization is different.

For each input $i = 0 \dots N_U - 1$, the input pattern consists of randomly drawn spike times $S_{ij} \in \mathcal{U}(0, t_{\text{trial}})$ with $j = 0 \dots N_{\text{stim}} - 1$, where $\mathcal{U}(0, t_{\text{trial}})$ is the uniform distribution on the

interval $[0, t_{\text{trial}}]$. All simulations use the same input spike times S_{ij} that are generated once to ensure comparability.

Weights for the random background have a uniform value w_B, so that every background spike causes the neuron to fire. Weights for input synapses are initialized to w_S, chosen so that single input spikes do not cause firing. See Table 2 for the numerical values.

2.2.2. Synaptic plasticity model

In the reward modulated STDP learning rule, the outcome of standard STDP drives so-called eligibility trace changes Δe_k:

$$\Delta e_k = \eta A_{\pm} \exp\left(-\frac{|\Delta t_k|}{\tau_{\pm}}\right) , \tag{5}$$

with learning rate η, time-difference between pre- and post-synaptic spike Δt_k for the k-th pair, STDP time constant τ_+ for pre-before-post pairings, τ_- for post-before-pre pairings, and, in the same fashion, amplitude parameters $A_{\pm}$. The Δe_k are accumulated on a per-synapse eligibility trace e. This trace decays exponentially with time constant τ_e:

$$e(t) = \sum_{\substack{k \\ t_k < t}} \Delta e_k \exp\left(-\frac{t - t_k}{\tau_e}\right) \tag{6}$$

with t_k being the time of the post-synaptic spike for pre-before-post pairings and of the pre-synaptic spike otherwise.

To calculate the weight update, a success signal S is used as modulating third factor. It represents the difference between reward received R and a running average of reward $\overline{R}$

$$S = R - \overline{R} \tag{7}$$

The reward is given at the end of each trial according to the learning task as defined in the next section. The running average is calculated as $\overline{R}_{n+1} = R_n + \left(R_n - \overline{R_n}\right)/5$ for the n-th trial. The weight update is then given by

$$\Delta = Se\left(t_{\text{trial}}\right) \tag{8}$$

with the trial duration t_{trial}.

In Fremaux et al. (2010) different time constants for pre-before-post ($\tau_+ = 20\,\text{ms}$) and post-before-pre ($\tau_- = 40\,\text{ms}$) are used. The amplitudes A_+ and A_- are chosen so that both parts are balanced, i.e. $A_+\tau_+ = -A_-\tau_-$. Synapses of the BrainScaleS wafer-scale system are designed for time constants of 20 ms. We do not want to assume, that this can be increased by a factor of two and therefore, we reduce τ_- to the same value as τ_+. Consequently we also use identical amplitudes to keep the STDP window W balanced. The plasticity rule described in this section represents the theoretical ideal model for our comparison that we refer to as the baseline model. Section 2.2.4 describes how this is mapped to hardware and the resulting constraints.

Table 1: Description of the network model used for the learning task after Nordlie et al. (2009). See Table 2 for numerical values of the parameters.

A: Model summary		
Populations	Three: input U, random background B, target T	
Connectivity	Feed-forward	
Neuron model	Leaky-integrate-and-fire, fixed voltage threshold, fixed absolute refractory period (voltage clamp)	
Synapse model	Exponentially shaped post-synaptic conductances	
Plasticity	Three-factor STDP	
Input	Fixed-length spike-trains with uniformly distributed firing times	
B: Populations		
Name	Elements	Population size
U	Stimulus generator	N_U
B	Poisson generator	N_B
T	LIF neurons	N_T
C: Connectivity		
Source	Target	Pattern
U	T	All-to-all, initial weights w_S
B	T	Non-overlapping 250 $\to$ 1, weight w_B
D: Neuron and synapse model		
Name	LIF neuron	
Type	Leaky integrate-and-fire, exponential-shaped synaptic conductances	
Sub-threshold dynamics	$\begin{cases} C_m \frac{dV}{dt} = g_L\left(E_L - V\right) + g(t)\left(E_e - V\right) & \text{if } t > t^* + \tau_{\text{ref}} \\ V(t) = V_{\text{reset}} & \text{else} \end{cases}$ $g(t) = w \exp\left(-t/\tau_{\text{syn}}\right)$	
Spiking	if $V(t-) < V_{\text{th}} \wedge V(t+) \geq V_{\text{th}}$ 1. set $t^* = t$ 2. emit spike with time-stamp t^*	
E: Plasticity		
Name	Three-factor STDP	
Spike pairing scheme	Reduced symmetric nearest-neighbor (Morrison et al., 2008)	
Weight dynamics	$\Delta = Sa(t)$ $a(t) = \sum_{\substack{i \\ t_i < t}} A_{\pm} \exp\left(\frac{\lvert\Delta t_i\rvert}{\tau_{\pm}}\right) \exp\left(-\frac{t-t_i}{\tau_e}\right)$ $w \in [w_{\text{min}}, w_{\text{max}}]$	
F: Input		
Type	Target	Description
Stimulus generator	U	N_{stim} spikes at random firing times distributed uniformly within the trial duration.
Poisson generators	B	Independent Poisson spike-trains with rate ν_B

Table 2: Numerical values for parameters. For parameter definitions see Table 1 and text.

Parameter	Value
N_U	250
N_B	$N_T \cdot 250$
N_T	5
C_m	500 pF
g_L	10 nS
E_L	-70 mV
E_e	0 mV
τ_{ref}	10 ms
V_{reset}	-60 mV
V_{th}	-50 mV
$A_\pm$	± 32 pS
$\tau_\pm$	20 ms
τ_e	$0.1 \ldots 1000$ s
w_{min}	0 nS
w_{max}	0.5 nS
w_B	20.0 nS
w_S	0.21 nS
$\hat{W}$	0.45 nS
ν_B	0.008 Hz
t_{trial}	1 s

2.2.3. Learning task

In reinforcement learning, reward given is determined by the nature of the learning task considered. In our case, the goal of the network is to reproduce a given target spike train. Hence, reward should be given in proportion to the similarity of the actual and target outputs, as measured by some metric. Here, we use a normalized version of the metric $D^{\text{spike}}[q]$ by Victor and Purpura (1996). $D^{\text{spike}}[q]$ represents the minimal cost of transforming the output of a trial into the target pattern by adding, deleting and shifting spikes. Adding and deleting have unit cost, while shifting by Δt has a cost of $q\Delta t$. For $\Delta t > 2/q$, deleting the spike and adding a new one at the correct time is cheaper than shifting it. Therefore, the parameter q controls the precision of the comparison. The cost parameter is set to $1/q = 20\,\text{ms}$ for our simulations.

Thus in a trial where neuron j fires with a spike train $X_{\text{out,j}}$ and the target was X_{target}, the contribution of neuron j to the reward is

$$R_j = 1 - \frac{D^{\text{spike}}[q]\,(X_{\text{out,j}}, X_{\text{target}})}{N_{\text{out,j}} + N_{\text{target}}}\,, \tag{9}$$

where $N_{\text{out,j}}$ and N_{target} are the number of spikes in $X_{\text{out,j}}$ and X_{target} respectively. Because $D^{\text{spike}}[q]$ is bound to $[0, N_{\text{out,j}} + N_{\text{target}}]$, R_j is limited to $[0, 1]$. The total reward R used for the weight update is the average of R_j over all N_T neurons.

The target spike train is generated by simulating the neural network with a set of reference weights W_{ij} for inputs $i = 0 \ldots N_U - 1$ and neurons $j = 0 \ldots N_T - 1$. All simulations use the same set of reference weights to ensure fair comparison:

$$W_{ij} = \begin{cases} \hat{W} \sin\left(\frac{i\pi}{N_U}\right) & \text{if } 0 \le i \le \frac{N_U}{2} \\ 0 & \text{if } \frac{N_U}{2} < i < N_U \end{cases} \tag{10}$$

with $\hat{W} = 0.45\,\text{nS}$. An example of an output spike pattern produced by the network is shown in Figure 3. A new target spike train is generated at the beginning of every simulation run. Its firing times can be different even for identical weights and stimulation, because of the random background stimulation.

2.2.4. Simulated hardware constraints

The baseline plasticity model described in Eqs 5-8 can not be reproduced exactly by the proposed system. This results in two distinct classes of effects: trade-offs introduced on purpose to reduce costs, for example in area, and non-ideal behavior of the hardware system.

In the first category, we analyze the effect of discretized weights and a limited access to analog variables by software running on the EPP. For the second category we study leakage in analog circuits and timing effects caused by finite processor speed and communication latencies.

Discrete weights In the hardware system, synaptic weights are discretized since they are stored as digital values in the synapse circuit. The number of bits per synapse is a critical

design decision when building a neuromorphic hardware system. Having fewer bits saves wafer area, so that more synapses can be implemented. More bits, on the other hand, allow for a higher dynamic range of the synaptic efficacies. The weight resolution also defines the minimum step size that can be taken by a learning rule. To analyze the sensitivity of learning performance to weight resolution, we modify the baseline model to use discrete weights with different numbers of bits. On a learning rule update, we precisely calculate the new weight (64 bit floating point) and round it to the nearest representable discrete weight value. The tie-breaking rule is round-to-even.

In the case of non-continuous weights with r bits, all updates with

$$|\Delta| < \frac{1}{2}\frac{w_{\text{max}} - w_{\text{min}}}{2^r - 1} \tag{11}$$

are discarded by rounding. Here w_{min} and w_{max} are the minimum and maximum weight values that can be represented and Δ is the true weight update (see Equation 8). Fewer bits per synapse means that more updates are discarded, causing the effective learning rule to increasingly deviate from the baseline learning rule.

A workaround to this problem is to perform discretized updates Δ_d probabilistically, depending on the exact weight update Δ as given by Equation 8. In this way, some of the updates that would otherwise be lost can be preserved. Using the correct update probabilities results in the average weight change being identical to that of the baseline model, i.e., without discretization.

To see this, we note that Δ_d can only assume values that are multiples of the discretization step $\delta_r = (w_{\text{max}} - w_{\text{min}}) \,/\, (2^r - 1)$, assuming $w_{\text{min}} = 0$. If the baseline weight change Δ is between the $(k-1)$-th and k-th step, the discrete update Δ_d is picked from those with probability $1 - p$ and p, respectively. Such a scheme leads to the average update $\langle \Delta_d \rangle$ for a given Δ being

$$\begin{aligned} \langle \Delta_d \rangle &= k\delta_r p + (k-1)\delta_r(1-p) && (12) \\ &= \delta_r(k-1) + \delta_r p\,. && (13) \end{aligned}$$

By picking p as

$$p = \frac{\Delta - (k-1)\delta_r}{\delta_r}\,, \tag{14}$$

it holds that $\langle \Delta_d \rangle = \Delta$.

Thresholded readout The eligibility trace is implemented using the analog accumulation in the synapse unit. For every spike pair, Equation 1 is evaluated and the corresponding eligibility trace change is added as charge on the local storage capacitors a_+ and a_- respectively. These values are not directly accessible to the EPP. Instead, using the evaluation unit described in Section 2.1.2 with threshold $\Theta = a_{\text{th}} - a_{\text{tl}}$, accumulation trace $a = a_+ - a_-$, configuration bits $e^+_{ac} = 1$, $e^+_{aa} = 1$, $e^+_{ca} = 0$, $e^+_{cc} = 0$ for the evaluation of b_+ and $e^-_{ac} = 0$, $e^-_{aa} = 0$, $e^-_{ca} = 1$, $e^-_{cc} = 1$ for b_-, the readout computes

$$b_\pm = \begin{cases} 1 & \text{if } \pm(a_+ - a_-) > \Theta \\ 0 & \text{otherwise} \end{cases}\,. \tag{15}$$

The weight update with threshold readout Δ_t is then performed using an update constant A

$$\Delta_t = SA\left(b_+ - b_-\right) . \tag{16}$$

The paraameters Θ and A should be chosen so as to minimize the deviation introduced by calculating weights according to Equation 16 instead of Equation 8. Ideally, one would like to satisfy $\langle\Delta_t\rangle = \Delta$. However, detailed analysis of the simulations (not shown) showed that the eligibility trace distributions for different synapses at different stages of learning were very different. In that context, choosing parameters Θ and A that minimize the difference between the baseline change Δ and the average effective change $\langle\Delta_t\rangle$ for a particular synapse would not in general have the same effect for other synapses. Instead, we resort to a heuristic method to fix global threshold and update constant, described below, and assess its effectiveness in simulations.

For the simulations presented here, a precursor run over 100 trials without learning was used to measure the final absolute eligibility value $\langle|a|\rangle$ averaged over all readout operations. The threshold Θ was then set to $\Theta^* = \langle|a|\rangle$ for the actual learning simulation. In this way, the average (across synapses) final eligibility value encountered during weight updates is close to the threshold. This represents a trade-off between exceeding the threshold only seldom, but then causing large – possibly disruptive – weight changes, and exceeding the threshold often, but only applying small changes.

With $N_p\left(\Theta\right)$ being the number of readout operations that exceed the threshold, i.e. b_+ or b_- are non-zero, and the total number of readout operations N, the update constant A is set to

$$A^* = \frac{N}{N_p(\theta^*)}\theta^* . \tag{17}$$

Thereby, the mean absolute eligibility value used with the readout $N_p(\Theta^*)A^*/N$ is effectively the same as $\langle|a|\rangle$ in the baseline model.

Analog drift The local accumulation units in the hardware synapses do not have a mechanism for controlled decay of the eligibility trace. An ideal implementation of the circuit would stay unchanged over time, after a spike-pair has caused an update. In reality there are leakage currents causing the accumulation traces a_+ and a_- and their difference a to drift. Leakage is caused by a number of processes that depend on transistor geometry, manufacturing process, temperature and internal voltages (Roy et al., 2003). It is therefore difficult to predict either time-scale, shape or variability of this effect. We try to get an estimate on the sensitivity of the model to uncontrolled temporal drift, by simulating learning with a drift function $\phi_i\left(t; a_0\right)$. Here t is the duration of the drift and a_0 is the starting value for $t = 0$. The index i is over all synapses and both trace polarities. This function describes the development of $a_+\left(t\right)$ and $a_-\left(t\right)$ between spike-pair induced updates. The accumulation value is given as the difference $a\left(t\right) = a_+\left(t\right) - a_-\left(t\right)$. We define an exponential drift function

$$\phi_i\left(t; a_0\right) = \begin{cases} a_0 e^{-\lambda_i t} & \text{for } \lambda_i > 0 \\ a_{\max} - \left(a_{\max} - a_0\right) e^{\lambda_i t} & \text{for } \lambda_i < 0 \\ a_0 & \text{else} , \end{cases} \tag{18}$$

where a_{max} is the maximum value that a_+ and a_- can assume and $\lambda_i = 1/\tau_{e,i}$ is the inverse time constant. Positive λ_i leads to exponential decay as it was used so far. Negative λ_i causes a drift away from zero, towards the limit a_{max}. For every synapse and for positive and negative traces, $\tau_{e,i}$ is drawn from a Gaussian distribution with mean τ_e and standard deviation $m_e\tau_e$ using the mismatch factor m_e. In the limit of large t, this allows for four final states of $a\,(t)$: Decay to zero, drift to a_{max} or $-a_{\text{max}}$ and remaining constant at a_0 for $\lambda_i = 0$.

It is important to note that we do not intend to precisely model the leakage behavior of the analog circuit. Instead, we use a simple model capturing the essence of drifting analog values to get an estimate for the sensitivity to this effect.

Delayed reward The hardware system is a physical model of the emulated network. Therefore, emulated time progresses continuously during network operation with the acceleration factor α relative to wall-clock time. During all communication and computation, network operation continues. The amount of reward for each trial is calculated by the control cluster, after the nominal trial duration has ended and output spike events have been transmitted to the cluster. The success signal is then determined and sent back to the embedded processor. Then, the plasticity program will sequentially execute the weight update for all synapses taking a certain amount of time per synapse. This time is consumed by the synapse array access and the weight computation.

These two effects are modeled by adding a constant delay D_R after the trial has finished and an update rate ν_s giving the number of updated synapses per second. The weight update for synapse i occurs at $t_i = t_{\text{trial}} + D_R + \frac{i}{\nu_s}$. The order in which synapses are updated is determined by their position in the synapse array and is therefore a result of the automated mapping process. For this study, we assume weight updates to be fast enough compared to the reward delay D_R and therefore use $t_i = t_{\text{trial}} + D_R$.

The delay causes a deviation from the ideal model because the accumulation capacitors a_+, a_- used to store the eligibility trace continue to decay. The eligibility value used for the weight update is then reduced by a factor

$$\beta = \exp\left(-\frac{D_R}{\tau_e}\right) \tag{19}$$

This can prevent a weight update that would have been made in the non-delayed case by reducing a below the readout threshold Θ. We assume that the delay D_R is known or can be estimated and lower the threshold to $\beta\Theta$.

In theory, this would allow to correct for arbitrary delay, since the exponential decay never reaches zero. In hardware this is not the case, because the eligibility readout is subject to noise. Therefore, after a certain delay, traces will be indiscernible from noise. To account for this, we simulate Gaussian distributed noise δa on the readout with standard deviation σ_a and mean 0. The value used for comparison to the threshold is then given by $a' = a + \delta a$. If a signal-to-noise ratio z^* is required for correct learning, a limit D_{max} for the delay can be calculated using the signal-to-noise ratio $z(t) = a(t)/\sigma_a$

$$z(t) \;=\; \frac{a\,(t_{\text{trial}})}{\sigma_a} \exp\left(-\frac{t - t_{\text{trial}}}{\tau_e}\right) \tag{20}$$

With $z\left(D_{\text{max}} + t_{\text{trial}}\right) = z^*$ and $a\left(t_{\text{trial}}\right) = a_{\text{max}}$, the maximally tolerable delay in the presence of noise is given by

$$D_{\text{max}} = -\tau_e \ln\left(\frac{z^*\sigma_a}{a_{\text{max}}}\right) \tag{21}$$

2.2.5. Measuring performance

Simulations consist of 10000 trials in 20 parallel runs with different random seeds. At the beginning of every run, 100 trials are simulated without learning: during this time the running average $\overline{R}$ can settle to a stable approximation of the reward. The average over R during these trials is used as the initial reward level R_{before} of this run. During the last 1000 trials of the simulation, it is assumed that learning has reached a stable state: the final reward level R_{after} is the average of R over these trials.

The model is simulated using the Brian simulator (Goodman and Brette, 2008). Weight updates are calculated with custom Python code using the NumPy package (Numpy, 2012).

3. Results

In the previous section, we analyzed a synaptic learning rule (Izhikevich, 2007; Florian, 2007; Fremaux et al., 2010), and the necessary adjustments that have to be made in order to implement it on a hardware system. The goal of this section is to quantify the sensitivity to constraints of the system – for example discretized weights or imperfections of analog circuits – to identify those critical for the model. Starting from the baseline configuration without hardware effects, we add constraints and measure their effect on the learning performance.

3.1. Baseline

The baseline model implements the learning rule described in Section 2.2 and Table 1 without hardware effects, and serves as comparison for simulations including such effects. The eligibility trace e of the theoretical model is identified with the local accumulation a in hardware synapses. Thereby, changes to the weight are deferred until the success signal S is given from the attached control cluster, after the produced spike train has been evaluated. New weights are assumed to be calculated using a software program running on the EPP.

The raster plot in Figure 3 shows the output spike train at several points in time during a learning simulation. In the beginning at trial 0, spikes are generated randomly by the background stimulation. Later on, the network learns to produce spikes at the targeted points of time indicated with red vertical bars. In the last trial, neurons fire close to most of the target times. The evolution of the reward obtained in each trial averaged over 20 runs is shown in Figure 5A. Variance in the last 1000 trials is due to the random background stimulation and to the exploratory behavior it generates in the learning rule. Most of the performance improvement is achieved within the first 2000 trials, the final level of reward being $R^{\text{base}}_{\text{after}} = 0.54 \pm 0.05$.

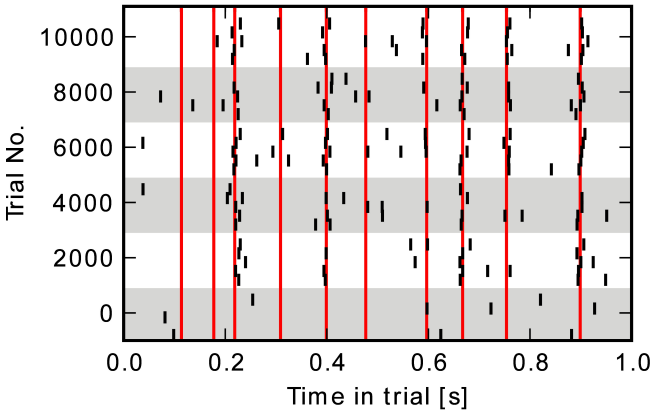

Figure 3: Raster-plot of output spike-events for all five neurons at intervals of 2000 trials. Red bars indicate the target firing times.

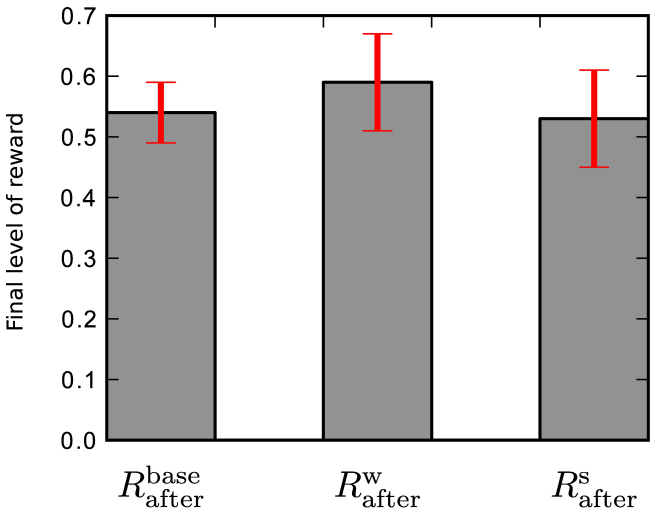

Figure 4: Final level of reward for: baseline simulation, randomized reference weights, and randomized stimulation pattern. The final performance level of the baseline simulation $R^{\mathrm{base}}_{\mathrm{after}}$ using reference weights W_{ij} and stimulation pattern S_{ij} is comparable to the final level of reward averaged over randomly chosen reference weights $R^{\mathrm{w}}_{\mathrm{after}}$ and stimulation patterns $R^{\mathrm{s}}_{\mathrm{after}}$.

This is the result using one particular set of reference weights W_{ij} and stimulation pattern S_{ij} that were defined in Section 2.2.1. To test how well this result generalizes to other weights and stimulation patterns we perform two additional experiments: first of all, we randomize the reference weights, so that in 20 simulation runs the network learns with a different set of reference weights in each run. These weights are drawn randomly from a uniform distribution, so that the k-th run uses reference weights $W_{ij}^k \in \mathcal{U}(w_{\text{min}}, w_{\text{max}})$ to generate its target spike train. This gives a final level of reward of $R_{\text{after}}^w = 0.59 \pm 0.08$ averaged over the 20 runs with different reference weights.

In the second experiment we again use the W_{ij} reference weights for all 20 simulations. The stimulation pattern is randomized by drawing new spike times for each run from a uniform distribution, so that the k-th run uses spike times $S_{ij}^k \in \mathcal{U}(0, t_{\text{trial}})$ for all trials. This gives a performance $R_{\text{after}}^s = 0.53 \pm 0.08$ averaged over the 20 different sets of stimulation patterns.

The final reward level for the baseline simulation, randomized reference weights and randomized stimulation pattern are shown in Figure 4. The data show, that the from here on used special case of reference weights W_{ij} and stimulation spike times S_{ij} is within the performance range of randomly selected reference weights and input spike timings. The variances on R_{after}^w and R_{after}^s also show that there is considerable variation in the unconstrained theoretical model. To reduce variation in our results, so that changes caused by hardware effects are more visible, we use W_{ij} and S_{ij} from here on.

3.2. Discretized weights

In designing the neuromorphic hardware system, one is faced with a trade-off between implementing more synapses with lower bit resolution and less synapses with higher resolution. Therefore, we would like to know how many bits are required for each synaptic weight to achieve good performance in the learning task. We perform a three-way comparison between the baseline model, a deterministic algorithm that simply rounds calculated weights to allowed representations and a probabilistic variant as outlined in Section 2.2.4. Using deterministic weight updates, all updates satisfying Equation 11 do not cause a weight change. With fewer bits more updates are lost and learning performance is expected to suffer. This is what can be seen in Figure 5. The simulations shown there compare performance of the baseline model, to a constrained model with discretized weights of decreasing resolution. Figure 5A also shows the full reward trace of a single run picked arbitrarily. The plot exhibits a number of sharp drops in reward that last only for a few trials, before returning to the previous performance level. The final level of performance is not affected by these glitches. For the 8 bit case, performance is as good as using continuous weights (Figures 5B). Figure 5C shows a slightly reduced performance for 6 bit. Using only 4 bit with deterministic updates causes performance to degrade: it does not reach the same final level of reward (Figure 5D black trace). Using probabilistic updates improves the performance for 4 bit close to the baseline level (Figure 5D green trace).

So in the task studied here, there is no gain in building synapses using more than 8 bit. Because weight updates are controlled by a programmable processor, it is possible to switch between deterministic and probabilistic updating even after the system has been manufactured. In this context, a trade-off can be made between number of synapses and reachable performance by

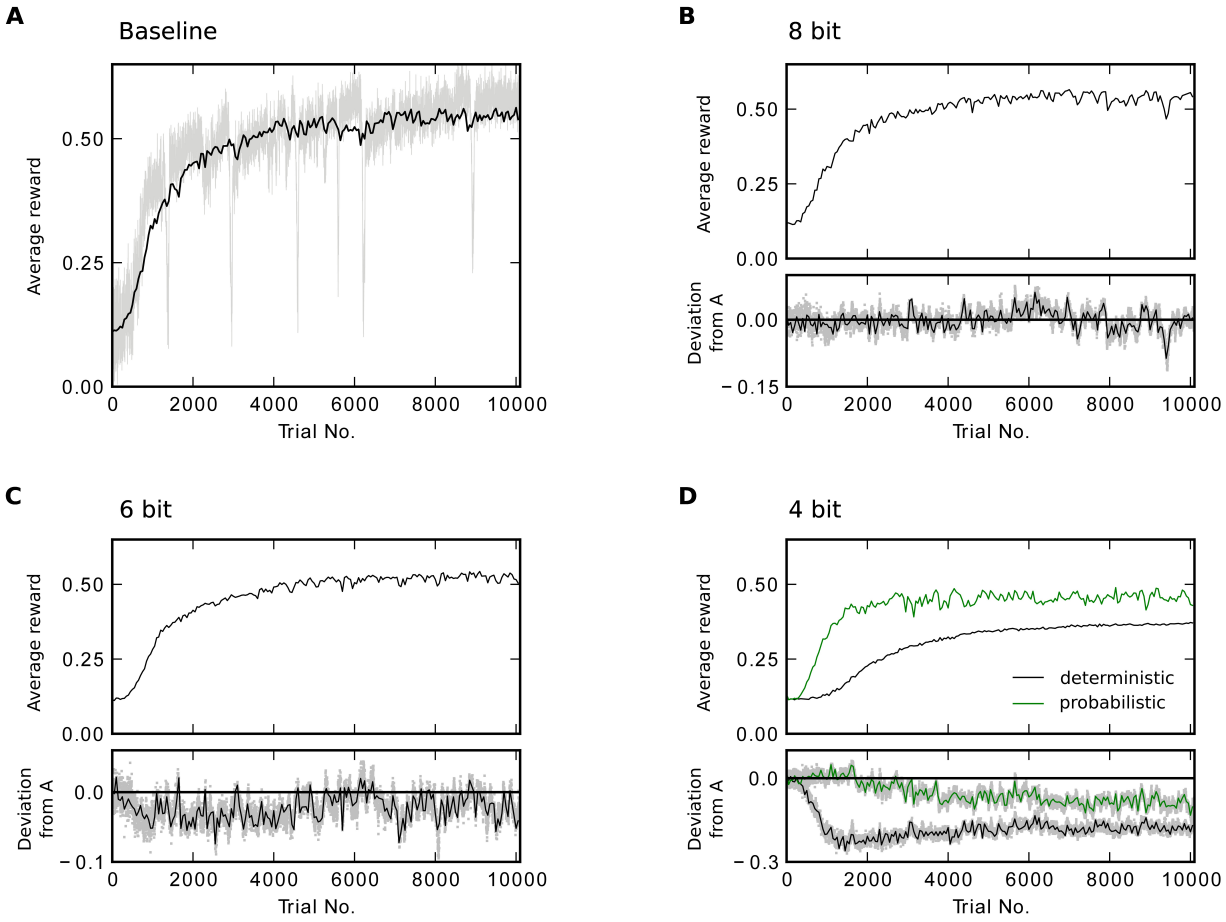

Figure 5: Reward traces showing the running average $\bar{R}$ (only every 50$^{\text{th}}$ point plotted) for different weight resolutions averaged over 20 runs. (A) Baseline performance with continuous weights. Additionally, the light gray trace shows the reward R for every trial of a single simulation. (B) Performance with 8 bit resolution. The lower plot shows the difference to the baseline model in (A). The shaded area shows the difference for every point in the trace instead of only for every 50$^{\text{th}}$. (C) Performance with 6 bit resolution. (D) Performance with 4 bit resolution. The black trace shows the result for deterministic updates. The green trace for probabilistic updates.

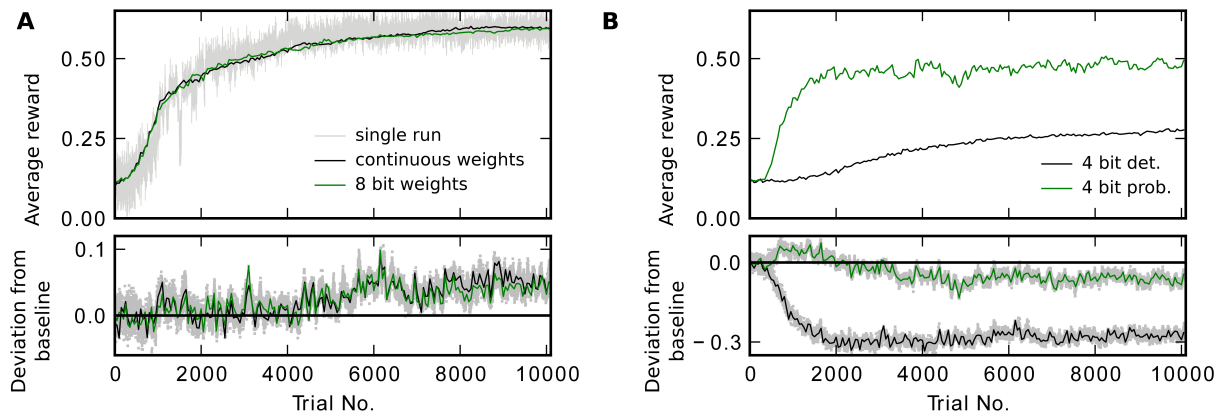

Figure 6: Performance with threshold readout. As in Figure 5 the running average of the reward $\bar{R}$ is plotted averaged over 20 runs. The lower plots show the difference to the baseline trace in Figure 5A. (A) Performance traces for continuous and 8 bit weights. In gray reward R for every trial in a single run with continuous weights is shown. (B) Performance traces for 4 bit resolution with deterministic and probabilistic updates.

using either probabilistic 4 bit or deterministic 8 bit synapses.

3.3. Thresholded readout

The hybrid approach of combining processor based digital computing with analog special-function units necessitates an interface between these two. At this interface some form of analog-to-digital conversion (ADC) has to take place. The simplest form of ADC is comparison to a threshold. We next ask whether such a simple interface is sufficient for good performance on the learning task. Figure 6 shows performance for different weight resolutions compared to baseline using the thresholded readout. In contrast to the simulations shown in Figure 5, updates are now calculated according to Equation 16 instead of Equation 8. In particular, Equation 16 does not directly use the eligibility trace $e(t_{\text{trial}})$, but the evaluation bits b_+, b_- determined by the readout mechanism (Equation 15). Performance in the case of continuous, 8 and 6 bit synapses (6 bit with threshold readout mechanism not shown) are nearly identical for both cases. When comparing traces for weights of the same resolution in Figures 5 and 6, those with threshold readout (Figure 6) show less variability between trials. For example, the trace of the single run in Figure 5A exhibits more noise than the one in Figure 6A. This is caused by the smoothing effect of the readout threshold, which effectively replaces extreme values of the eligibility trace $e(t_{\text{trial}})$ with the update constant $A = A^*$. The update constant A^* is determined heuristically according to Equation 17. The data show a small improvement in final performance over baseline for continuous and 8 bit weights, when using the readout threshold. This can be explained by the reduced noise on the reward trace that allows for closer approximation of optimal weights. When using probabilistic updates (Figure 6B, green trace), 4 bit are enough to come close to baseline performance. With deterministic updates and 4 bit synapses, performance is even worse using

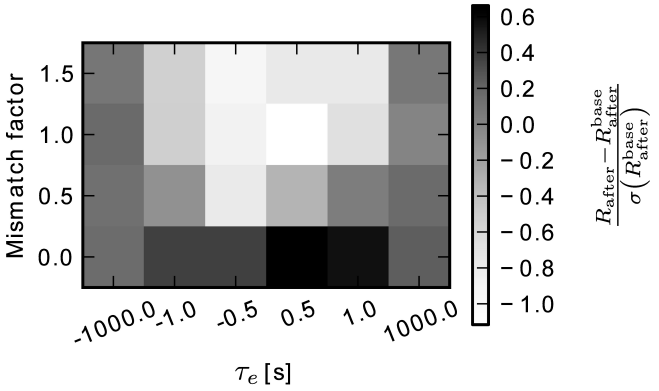

Figure 7: Difference of final reward to the baseline simulation $R_{\text{after}} - R^{\text{base}}_{\text{after}}$ in units of the baseline standard deviation. The varied parameters are the average time constant and the amount of mismatch between synapses.

threshold readout than without (black traces in Figures 5D and 6B).

Hence the simple readout method consisting in using only a threshold comparison does not reduce performance. Therefore, the qualitative result from the previous section still holds: with deterministic updates 6 bit is enough for satisfying performance. If updates are performed in a probabilistic manner, 4 bit is sufficient.

3.4. Analog drift

In the hardware system, the eligibility trace is implemented as an analog variable inside the synapse circuit. It is therefore subject to drift caused by leakage currents. In Equation 18, we have proposed to model this using a drift function. Additionally, this behavior varies between synapses due to imperfections introduced by the manufacturing process. This is taken account for by randomly drawing parameters for the drift function from a Gaussian distribution.

To assess the impact of this drift on the performance in the learning task, we performed a sweep over a number of average time constants and degrees of mismatch between synapses. The results of the simulation, using continuous weights and the thresholded eligibility readout described above, are shown in Figure 7. The gray value indicates the difference between R_{after} and the baseline value $R^{\text{base}}_{\text{after}}$ (Section 3.1) in units of the standard deviation of the baseline simulation (darker color is better). All values fall within one standard deviation of the baseline case, which means that performance is only weakly sensitive to changes of time constant and mismatch of the eligibility trace. The best performance is achieved for $\tau_e = 0.5\,\text{s}$ and no mismatch. This is equivalent to the black trace in Figure 6A, which also shows slightly improved performance to baseline. The improvement is explained by the smoothing effect of the threshold readout as discussed in Section 3.3. For very large time constants, i.e. $\tau_e = \pm 1000\,\text{s}$, drift is negligible compared to the trial duration $t_{\text{trial}} = 1\,\text{s}$. This leads to minor deviations in the leftmost and rightmost columns of Figure 7. The worst performance is obtained for small time constants τ_e with large mismatch factor m_e, because for τ_e lesser than or equal to the trial duration, the effect

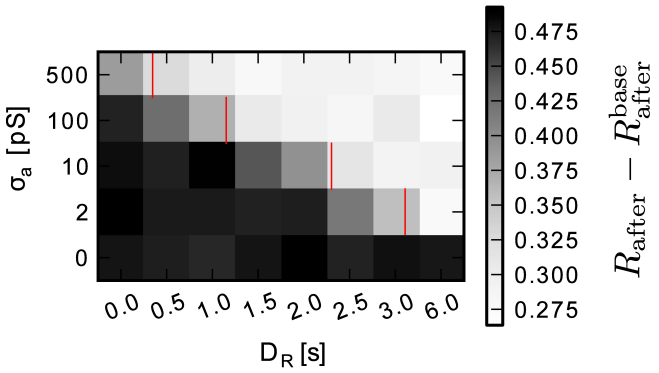

Figure 8: Improvement in reward $R_{\text{after}} - R_{\text{before}}$ by learning for a range of delays and accumulator readout noise levels. Red bars indicate the predicted maximally tolerable delay (Equation 21). Data is averaged over 15 simulation runs.

of drift is more important.

In this test, the model has shown to be robust to large deviations from the temporal behavior of the eligibility trace in the baseline model. Drift towards the positive and negative extrema of the eligibility trace, which is the opposite of the desired decaying behavior, does not affect performance. Neither does variation of up to 150 % of the time constant. This shows the model to be a well-suited candidate for implementation in neuromorphic hardware, where large variations and distortions are often encountered.

3.5. Delayed reward

In the proposed system, the simulation of the neural network is carried on by analog hardware elements, while the simulation of the environment is left to a conventional computer system. In this context, latencies due to technical reasons – e.g., by communication with the environment or computation by the EPP– can cause temporal delays with respect to ideal calculations. Additionally, the analog readout of the accumulation traces a_+, a_- is affected by noise.

To better understand the impact of these effects on learning performance, a sweep over readout noise and reward latency values was performed, the results of which are shown in Figure 8. The simulation did not include mismatched drift, but used a fixed time constant of 500 ms with continuous weights. The gray value represents the improvement in reward by learning $R_{\text{after}} - R_{\text{before}}$. The data shows that depending on the amount of noise learning is impaired by the delay. The red bars indicate the predicted maximally tolerable delay assuming a signal-to-noise ratio of one is required (Equation 21). The simulation fits the prediction well. A noise level of $\sigma_a = 500\,\text{pS}$ corresponds to 50 % of the maximum of the eligibility trace a_{max}.

The simulation results confirm that noise on the local accumulation circuit limits tolerable delay. Because of the accelerated time base of the system, communication delays can easily reach seconds of emulated time. With an acceleration factor of $\alpha = 10^5$ one second of emulated time is equivalent to $10\,\mu$s. So with 1 % of noise ($\sigma_a = 10\,\text{pS}$), the round-trip-time to the environment must be less than $20\,\mu$s for a $\tau_e = 500\,\text{ms}$ time constant. Equation 21 can be used to find working

combinations of the parameters round-trip-time, analog noise and time constant.

4. Discussion

In this study we have proposed a hybrid architecture for plasticity, combining local analog computing with global, program-based processing. We have then simulated a reward-modulated spike-timing-dependent plasticity learning rule studied by Fremaux et al. (2010) to analyze its implementability. Starting from a baseline case with no hardware effects, the level of hardware detail of the simulations was increased, with a focus on the negative effects introduced by an implementation using the proposed system. Note that we did not try to precisely model the hardware device, as it would be done, for example, in a transistor level simulation. Instead, our goal was to find the effects to which the model is sensitive in order to guide future design decisions.

Overall, we did not find major obstacles for the proposed implementation, but we showed that some design choices are critical to the proper functioning of the learning rule. In the following, we will discuss guidelines concerning weight resolution, implementation of the eligibility trace and the importance of low-latency communication. After that, we will compare the design with other hardware systems and discuss the limitations of this study.

4.1. Weight resolution

For neuromorphic hardware systems using digitally represented weights, a key question is how many bits to use per synapse, as this determines the amount of wafer area the circuit requires. For networks with highly connected neurons, small synapses are important for scalability. This drives implementations to a reduction of the number of bits used for the weight compared to software simulators, which typically use a quasi-continuous 32 or 64 bit floating-point representation. On the other hand, on-line synaptic plasticity learning rules, for example STDP, require incremental changes to the weights. Discretization confines these changes to a grid with a resolution determined by the number of bits.

For the synaptic plasticity model and the learning task considered, we found that this indeed limits learning performance when using deterministic updates and 4 bit weights. Two solutions to this problem were tested: using higher resolutions and making updates probabilistically. In the former case, a performance comparable to the continuous case is reached with 6 bit. With probabilistic updates, the performance of 4 bit synapses could be improved to nearly the same level. Therefore, it is not necessary to build high resolution hardware synapses comparable to software simulators, but even a modest number of bits gives good performance.

In Seo et al. (2011) the authors arrive at a similar result. They built a completely digital system in a version with 1 bit synapses and probabilistic updates and one with 4 bit synapses and deterministic updates. Learning performance in a benchmark task is improved in the latter case, but adds additional costs in area and power consumption.

In Pfeil et al. (2012) the question of weight resolution was also studied for the BrainScaleS wafer-scale system using a synchrony detection task. Comparable to our findings, they report

8 bit weights to perform as good as floating-point weights. 4 bit weights were sufficient for solving the task, but did not reach the same performance.

4.2. Implementation of the eligibility trace

In neural models of reinforcement learning, the eligibility trace serves an important purpose: it allows to connect neural activity with reward. Reward typically arrives with a delay with respect to the activity underlying causing actions respective spikes. But only when reward arrives does the agent know how to change the weights. The hybrid concept of local analog accumulation and global processor-based weight computation fits this model very well. Therefore, we can identify the local circuit in the synapse with the eligibility trace. However there are two differences. First, the processor does not have direct access to the accumulated value, but can only do a simple comparison operation (Equation 2). Second, there is no controlled exponential decay of the accumulator. The analysis in Sections 3.3 and 3.4 shows no degradation in learning performance by both effects. On the other hand, the lack of controlled and possibly configurable decay presents a constraint to the fidelity, with which learning rules can be implemented. It is not clear, how other learning tasks would be affected by this lack.

4.3. Impact of real-world timings

In the presence of delayed reward, three parameters govern whether learning is possible: 1) communication round-trip-time to the environment and back, 2) the amount of noise on the eligibility trace, and 3) the time constant of decay of the eligibility trace. Equation 21 allows to determine working combinations of them. Reducing the speed-up factor would make communication latency less of a problem, but it would require longer lasting analog storage to achieve the same time constant in emulated time. Small long-term analog memory is difficult to build due to leakage effects. Therefore, the triangle of parameters needs to be carefully balanced. A different approach to deal with communication latency would be to execute the environment on the EPP itself. This would require adding direct access to spike times by the processor.

4.4. Comparison to other STDP implementations

Plasticity implementations found in the literature typically focus on variants of unsupervised STDP and use fixed-function hardware. For example in Indiveri et al. (2006) STDP works on bi-stable synapses and is implemented using fully analog circuits. In Ramakrishnan et al. (2011) analog floating-gate memory is used for weight storage that can be subjected to plasticity. In contrast, Seo et al. (2011) describes a fully digital implementation using counters and linear-feedback shift registers for probabilistic STDP with single-bit synapses. All three systems perform weight updates immediately for individual spike pairs with a fixed algorithm. This rules out the ability to implement an eligibility trace to solve the distal reward problem in reinforcement learning (Izhikevich, 2007).

However, there are systems that also use a general-purpose processor for plasticity. For example, in Vogelstein et al. (2003) an implementation of STDP in an address-event representation

(AER) routing system is presented. They use three individual chips: a custom integrate-and-fire neuron array, an SRAM based look-up table for synaptic connections and a micro-controller for plasticity. For STDP, the micro-controller processes every spike and maintains queues of pre- and post-synaptic events. This necessitates multiple off-chip memory accesses for every event and at regular time steps. Contrary to our approach, their system has access to the detailed timing of spikes and can therefore additionally implement rules including short-term effects, as in Froemke et al. (2010). However, in terms of scalability, our proposed system is superior due to the integration of processor, event routing and neuronal dynamics onto the same wafer. This reduces power consumption by eliminating communication across chip boundaries. Also, due to the hybrid architecture of analog accumulation and digital weight computation, the workload for the processor is reduced. This is an important aspect if a high speed-up factor is aimed for.

The system reported in Davies et al. (2012) is a specialized multi-processor platform for neural simulations. In implementing STDP, a key constraint for them is limited access to weights stored in external memory. They solve this problem by predicting firing times based on the membrane potential. This simultaneously illustrates the strength and weakness of this architecture. Since the system is completely digital, they have unconstrained access to state variables, such as the membrane potential. With analog neurons, this always requires some form of analog to digital conversion. On the other hand, weights are stored external to the processor and have to be transfered between chips. In our system, close integration of weight memory and processor on the same substrate in addition to the optimized input/output instructions of the SYNAPSE special-function unit, make weight access more efficient.

In conclusion, the hybrid processor based architecture proposed in this study represents a novel plasticity implementation for hardware. To our knowledge, it introduces two novel concepts: first, the integration of a general-purpose processor for plasticity onto the neuromorphic substrate, and second, the close interaction with specialized analog computational units using an extension of the instruction set. In combination, this allows for reward-based spike-timing-dependent synaptic plasticity in reinforcement learning tasks.

4.5. Limitations

The goal of this study was to analyze the implementability of a reinforcement learning task on a proposed novel hardware system. The technical implementability of the system itself was not subject of this study. We assumed a sufficiently fast processor for the delay analysis (Section 2.2.4). It should be part of the design process of a future implementation to test performance against our simulations. The updating speed could limit the amount of plastic synapses per processor depending on the decay time constant τ_e. We also did not model the analog part of the system in detail, but restricted simulations to a generic drift function. Measurements in the existing BrainScaleS wafer-scale system could be used to characterize the drifting behavior. However, considering that we did not see degraded performance over a large range of time constants and fixed-pattern variation, it does not seem likely that performance would be worse in a more accurate model.

With regard to the model tested here, we restricted the study to one specific task of spike train learning, which is a generic and general learning task for spiking neurons: many tasks can be

formulated as a relaxed version of spike train learning. We showed that the performance of the model is not negatively affected by hardware constraints. It remains an open question whether there are other tasks that give good performance in software simulations, but fail when hardware constraints are included. We restricted the study to epochal learning with defined trial-duration ended by the application of the reward. In a next step, this approach should be extended to continuous time learning scenarios. In this case, processor update speed and the size of the decay time constant could play a more important role.

Conflict of Interest Statement

The authors declare that the research presented in this paper was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Acknowledgments

The research leading to these results has received funding by the European Union 7th Framework Program under grant agreement nos. 243914 (Brain-i-Nets) and 269921 (BrainScaleS). We would like to thank Thomas Pfeil for helpful discussions.

References

John Backus. Can programming be liberated from the von neumann style?: a functional style and its algebra of programs. *Commun. ACM*, 21(8):613–641, August 1978. ISSN 0001-0782. doi: 10.1145/359576.359579. URL http://doi.acm.org/10.1145/359576.359579.

R. Brette and W. Gerstner. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *J. Neurophysiol.*, 94:3637 – 3642, 2005. doi: NA.

Natalia Caporale and Yang Dan. Spike timing-dependent plasticity: A hebbian learning rule. *Annual review of neuroscience*, February 2008. ISSN 0147-006X. doi: http://dx.doi.org/10.1146/annurev.neuro.31.060407.125639.

S. Davies, F. Galluppi, A.D. Rast, and S.B. Furber. A forecast-based stdp rule suitable for neuromorphic implementation. *Neural Networks*, 32(0):3 – 14, 2012. ISSN 0893-6080. doi: 10.1016/j.neunet.2012.02.018. URL http://www.sciencedirect.com/science/article/pii/S0893608012000470. Selected Papers from IJCNN 2011.

A. P. Davison, D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger. PyNN: a common interface for neuronal network simulators. *Front. Neuroinform.*, 2(11), 2008.

M. Ehrlich, K. Wendt, L. Zühl, R. Schüffny, D. Brüderle, E. Müller, and B. Vogginger. A software framework for mapping neural networks to a wafer-scale neuromorphic hardware system. In *Proceedings of the Artificial Neural Networks and Intelligent Information Processing Conference (ANNIIP) 2010*, pages 43–52, 2010.

Michael A. Farries and Adrienne L. Fairhall. Reinforcement learning with modulated spike timingdependent synaptic plasticity. *Journal of Neurophysiology*, 98(6):3648–3665, December 2007. doi: 10.1152/jn.00364.2007. URL http://jn.physiology.org/content/98/6/3648.abstract.

J. Fieres, J. Schemmel, and K. Meier. Realizing biological spiking network models in a configurable wafer-scale hardware system. In *Proceedings of the 2008 International Joint Conference on Neural Networks (IJCNN)*, 2008.

Rzvan V. Florian. Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Computation*, 19(6):1468–1502, April 2007. ISSN 0899-7667. URL http://dx.doi.org/10.1162/neco.2007.19.6.1468.

Nicolas Fremaux, Henning Sprekeler, and Wulfram Gerstner. Functional requirements for reward-modulated spike-timing-dependent plasticity. *The Journal of Neuroscience*, 30:13326–13337, 2010.

Robert C Froemke, Dominique Debanne, and Guo-Qiang Bi. Temporal modulation of spike-timing-dependent plasticity. *Frontiers in Synaptic Neuroscience*, 2(19), 2010. ISSN 1663-3563. doi: 10.3389/fnsyn.2010.00019. URL http://www.frontiersin.org/synaptic_neuroscience/10.3389/fnsyn.2010.00019/abstract.

Steve B. Furber, David R. Lester, Luis A. Plana, Jim D. Garside, Eustace Painkras, Steve Temple, and Andrew D. Brown. Overview of the SpiNNaker system architecture. *IEEE Transactions on Computers*, 99(PrePrints), 2012. ISSN 0018-9340. doi: http://doi.ieeecomputersociety.org/10.1109/TC.2012.142.

Wulfram Gerstner and Werner Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

Dan Goodman and Romain Brette. Brian: a simulator for spiking neural networks in Python. *Front. Neuroinform.*, 2(5), 2008.

G. Indiveri, E. Chicca, and R. Douglas. A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity. *IEEE Transactions on Neural Networks*, 17(1):211–221, Jan 2006.

Eugene M. Izhikevich. Solving the distal reward problem through linkage of stdp and dopamine signaling. *Cerebral Cortex*, 17(10):2443–2452, 2007. doi: 10.1093/cercor/bhl152. URL http://cercor.oxfordjournals.org/content/17/10/2443.abstract.

R. Legenstein, D. Pecevski, and W. Maass. A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Computational Biology*, 4(10):e1000180, 2008.

P. Livi and G. Indiveri. A current-mode conductance-based silicon neuron for address-event neuromorphic systems. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pages 2898 –2901, 24-27 2009. doi: 10.1109/ISCAS.2009.5118408.

Sebastian Millner, Andreas Grübl, Karlheinz Meier, Johannes Schemmel, and Marc-Olivier Schwartz. A VLSI implementation of the adaptive exponential integrate-and-fire neuron model. In J. Lafferty et al., editors, *Advances in Neural Information Processing Systems 23*, pages 1642–1650, 2010.

Abigail Morrison, Markus Diesmann, and Wulfram Gerstner. Phenomenological models of synaptic plasticity based on spike timing. *Biological Cybernetics*, 98(6):459–478, June 2008. ISSN 0340-1200. doi: 10.1007/s00422-008-0233-1.

Eilen Nordlie, Marc-Oliver Gewaltig, and Hans Ekkehard Plesser. Towards reproducible descriptions of neuronal network models. *PLoS Comput Biol*, 5(8):e1000456, 08 2009. doi: 10.1371/journal.pcbi.1000456. URL http://dx.doi.org/10.1371%2Fjournal.pcbi.1000456.

Numpy. Website. http://numpy.scipy.org, 2012.

Thomas Pfeil, Tobias C Potjans, Sven Schrader, Wiebke Potjans, Johannes Schemmel, Markus Diesmann, and Karlheinz Meier. Is a 4-bit synaptic weight resolution enough? - constraints on enabling spike-timing dependent plasticity in neuromorphic hardware. *Frontiers in Neuroscience*, 6(90), 2012. ISSN 1662-453X. doi: 10.3389/fnins.2012.00090.

Wiebke Potjans, Markus Diesmann, and Abigail Morrison. An imperfect dopaminergic error signal can drive temporal-difference learning. *PLoS Comput Biol*, 7(5):e1001133, 05 2011. doi: 10.1371/journal.pcbi.1001133. URL http://dx.doi.org/10.1371%2Fjournal.pcbi.1001133.

PowerISA. PowerISA version 2.06 revision b. Technical report, power.org, July 2010. Available at http://www.power.org/resources/reading/.

S. Ramakrishnan, P.E. Hasler, and C. Gordon. Floating gate synapses with spike-time-dependent plasticity. *Biomedical Circuits and Systems, IEEE Transactions on*, 5(3):244–252, 2011.

Kaushik Roy, Saibal Mukhopadhyay, and Hamid Mahmoodi-Meimand. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer cmos circuits. In *Proceedings of the IEEE*, volume 91, pages 305 – 327. IEEE, February 2003.

J. Schemmel, A. Grübl, K. Meier, and E. Muller. Implementing synaptic plasticity in a VLSI spiking neural network model. In *Proceedings of the 2006 International Joint Conference on Neural Networks (IJCNN)*. IEEE Press, 2006.

J. Schemmel, D. Brüderle, K. Meier, and B. Ostendorf. Modeling synaptic plasticity within networks of highly accelerated I&F neurons. In *Proceedings of the 2007 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 3367–3370. IEEE Press, 2007.

J. Schemmel, J. Fieres, and K. Meier. Wafer-scale integration of analog neural networks. In *Proceedings of the 2008 International Joint Conference on Neural Networks (IJCNN)*, 2008.

J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proceedings of the 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1947–1950, 2010.

Stefan Scholze, Stefan Schiefer, Johannes Partzsch, Stephan Hartmann, Christian Georg Mayr, Sebastian Höppner, Holger Eisenreich, Stephan Henker, Bernhard Vogginger, and Rene Schüffny. VLSI implementation of a 2.8GEvent/s packet based AER interface with routing and event sorting functionality. *Frontiers in Neuromorphic Engineering*, 5(117):1–13, 2011.

J. Seo, B. Brezzo, Y. Liu, B.D. Parker, S.K. Esser, R.K. Montoye, B. Rajendran, J.A. Tierno, L. Chang, D.S. Modha, and D.J. Friedman. A 45nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. In *Custom Integrated Circuits Conference (CICC), 2011 IEEE*, pages 1 –4, sept. 2011. doi: 10.1109/CICC.2011.6055293.

James E. Smith. A study of branch prediction strategies. In *25 years of the international symposia on Computer architecture (selected papers)*, ISCA '98, pages 202–215, New York, NY, USA, 1998. ACM. ISBN 1-58113-058-9. doi: 10.1145/285930.285980. URL http://doi.acm.org/10.1145/285930.285980.

J.E. Smith and A.R. Pleszkun. *Implementation of precise interrupts in pipelined processors*, volume 13. IEEE Computer Society Press, 1985.

Richard Stallman. *Using the GNU Compiler Collection*. Free Software Foundation, 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA, for gcc version 4.7.2 edition, 2012. URL http://gcc.gnu.org.

R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.

J. D. Victor and K. P. Purpura. Nature and precision of temporal coding in visual cortex: a metric-space analysis. *J Neurophysiol*, 76(2):1310–1326, 1996.

R. Jacob Vogelstein, Francesco Tenore, Ralf Philipp, Miriam S. Adlerstein, David H. Goldberg, and Gert Cauwenberghs. Spike timing-dependent plasticity in the address domain. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 1147–1154, Cambridge, MA, 2003. MIT Press.

Karsten Wendt, Matthias Ehrlich, and René Schüffny. A graph theoretical approach for a multistep mapping software for the facets project. In *CEA'08: Proceedings of the 2nd WSEAS International Conference on Computer Engineering and Applications*, pages 189–194, Stevens Point, Wisconsin, USA, 2008. World Scientific and Engineering Academy and Society (WSEAS). ISBN 978-960-6766-33-6.

Jayawan H.B. Wijekoon and Piotr Dudek. Compact silicon neuron circuit with spiking and bursting behaviour. *Neural Networks*, 21(2-3):524 – 534, 2008. ISSN 0893-6080. doi: DOI:10.1016/j.neunet.2007.12.037. URL http://www.sciencedirect.com/science/article/B6T08-4RFSCV3-5/2/c005fcc0c2482bf724210a079932484e. Advances in Neural Networks Research: IJCNN '07, 2007 International Joint Conference on Neural Networks IJCNN '07.